C Programing

- Four Rules for Better C Programming
  - Use strict ANSI C constructs
  - Never use global variables
  - Fix all compiler warnings - will often be type warnings
  - Use prototypes for all subroutines

C Programing

- Use strict ANSI C constructs

  All C compilers allow for enforcement of strict ANSI C. You should use these options at all times. With the gcc compiler under the Linux operating system, the compiler flags are
  gcc -ansi -Wall -pedantic
  These flag settings will insure that you receive warnings for any poor programming constructs in your code.

C Programing

- Never use global variables

  Global variables should never be used. There are exceptions to this rule, but you will never face a situation where they are truely needed. Global variables will lead to “spagetti code”, that can not be understood or debugged.

C Programing

- Fix all compiler warnings

  Any warning from the compiler means that you have an error in you program or are using poor programming style. Either problem is serious and should be corrected. Typically, warnings are generated when variables are not properly retyped. The only exception to this rule is when you dynamically allocate memory. In this case, you may receive a warning about the byte location.

C Programing

- Use prototypes for all subroutines

  Every subroutine you write should have a prototype that defines its arguement types. This prototype should generally be contained in a single header file with a name such as *defs.h*.

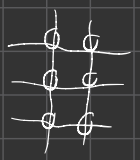
Memory

- memory is linear
- 8 bits → bite
- 4 bites → int
- 8 bytes → double precision

p[0] p[1]

- 2^{31} → amount of memory in 32-bit system?
- $2^{10} = 1024 = 1k$
- $2^{20} = 1M$
- $2^{30} = 1G$
  ↳ $2^{31} = 2G$

- be careful to not index outside of allocated memory
- accessing memory outside of your space - segmentation fault
  ENIAC computer
- core memory - magnetizing individual ferrous metal rings



int *p;
- malloc and calloc commands

char *p; → -128 to 128
unsigned char *p → 0 to 255

p* = calloc(N, sizeof(char))
p* = @(float *p) pT - re-type pointer to float
a = p[0]; → bus error

- memory errors in C are nasty to debug → errors exhibit themselves in very different places than where the true problem originates

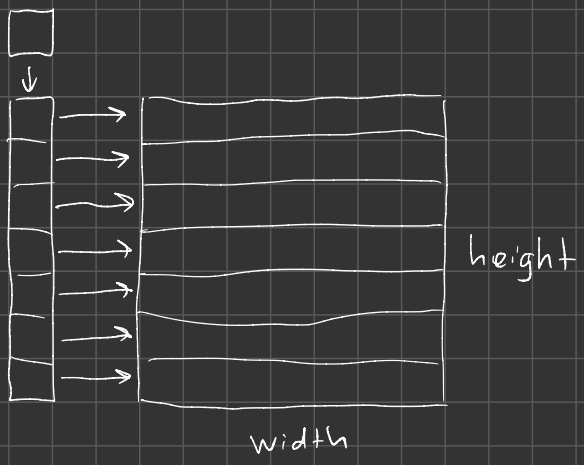
- memory leaks - deadly

2/4/21

img = calloc(width * height * sizeof(...))

- caching → getting blocks of memory at once
- allocate memory → allocate RAM in computer
- flops - floating point operations per second

- multialloc()



if width is small compared to height, overhead associated with pointers is relatively cumbersome